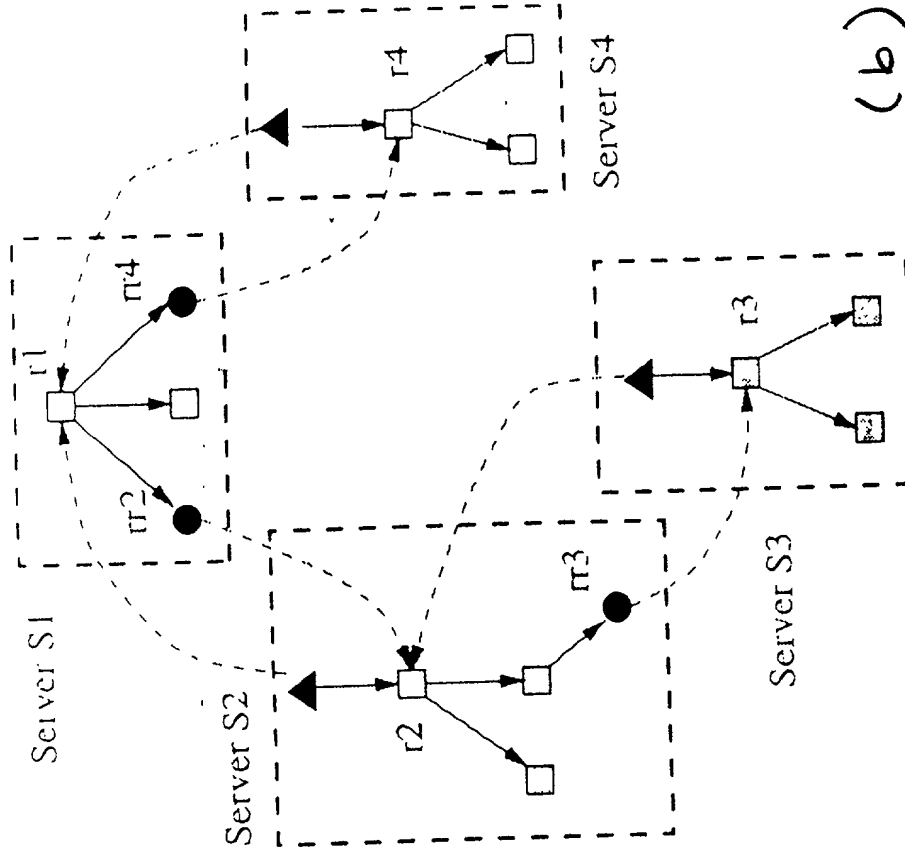
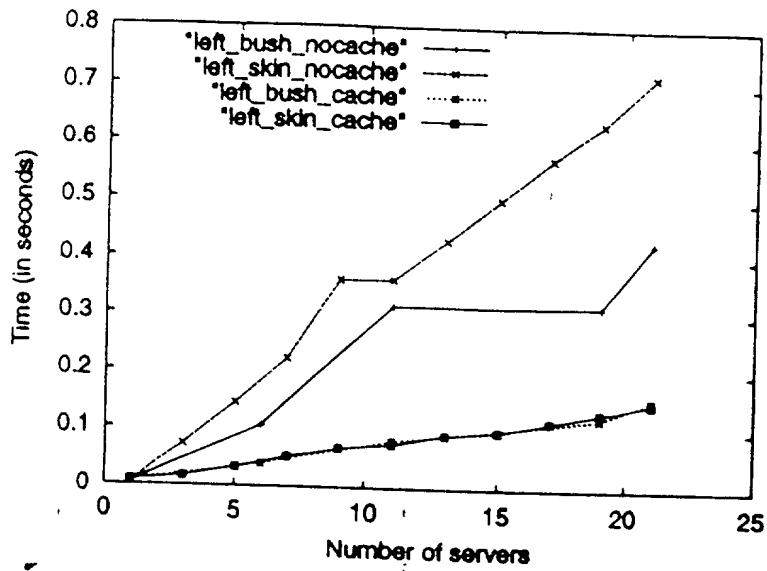
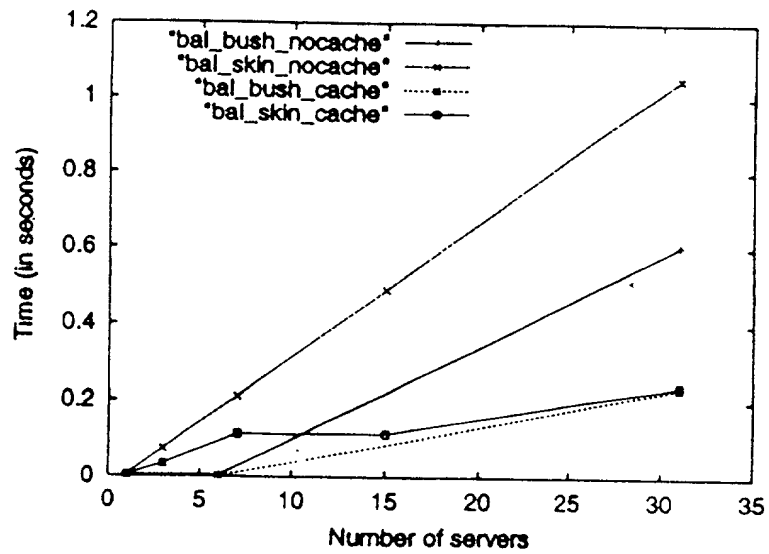


FIG. 1



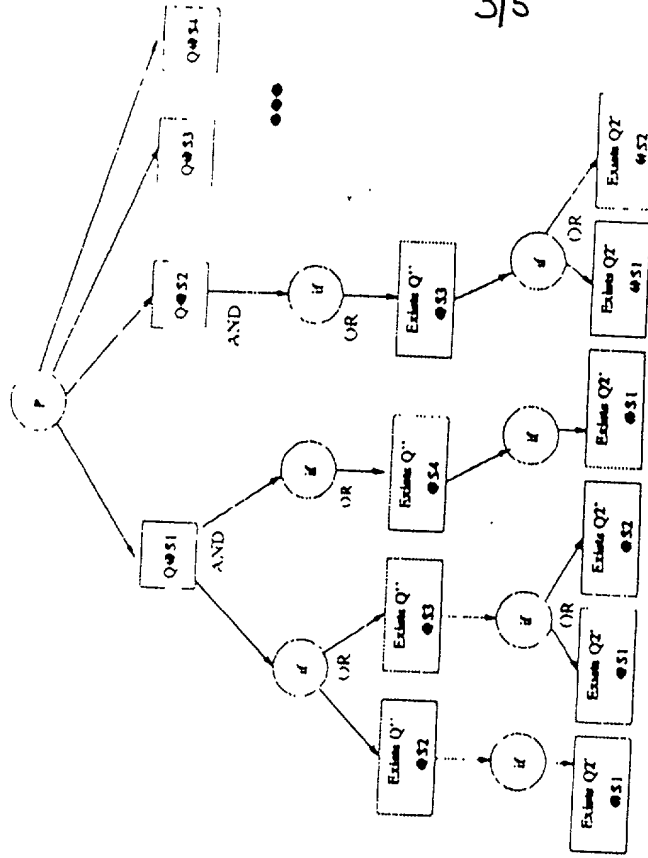


(a) Left-deep Topologies



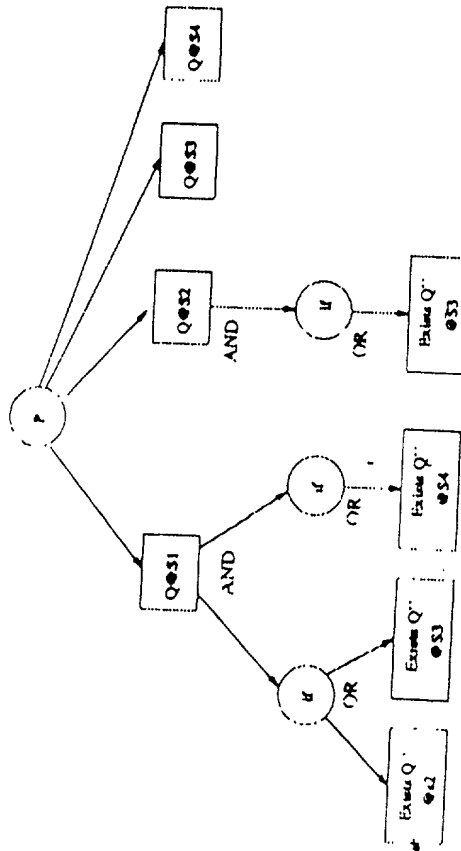
(b) Balanced Topologies

FIG. 2



$$Q = (d \ Q' \ (a \ Q1' \ Q2'))$$

FIG. 4



$$Q = (d \ Q' \ Q'')$$

FIG. 3

```

Algorithm Schedule(PT) {
  Answer := { }; Pending := { }; Enabled := { };
  for each n in leaves(PT) do computeQueryNode(n);
  while (Enabled ≠ { } OR Pending ≠ { })
    L := chooseForSchedule(Enabled); /* implements a particular scheduling policy */
    for each (Q, S) in L do
      Pending := Pending ∪ {(Q, S)}; LDAP_issueQuery(Q, S);
      LDAP_waitForEvent(e);
      case e.type of
        boolean answer for Q@S: Pending := Pending − {(Q, S)};
          storeCache(Q, S, e.value);
          for n in getCacheWaitinglist(Q, S) do {
            n.value := e.value;
            computeConditionalNode(n.parent); }
          directory entry for Q@S: Answer := Answer ∪ {e.value};
          End-of-Entries for Q@S: Pending := Pending − {(Q, S)};
      return Answer;
}

function computeQueryNode(n) {
  if all n's children are computed then
    Q := generateQueryExpression(n.Query); /* expands all if-macros */
    S := n.Server; v := getCache(Q, S);
    case v of
      INEXISTENT:      insertCache(Q, S, PENDING);
                       Enabled := Enabled ∪ {(Q, S)};
                       addCacheWaitingList(Q, S, n);
      PENDING:         addCacheWaitingList(Q, S, n);
      TRUE, FALSE:     n.value := v;
                       computeConditionalNode(n.parent)
    }
}

function computeConditionalNode(n) {
  if (exists p in n.children such that p.value = TRUE) then
    n.value := TRUE; computeQueryNode(n.parent);
  else if (all n's children are computed) then
    n.value := FALSE; computeQueryNode(n.parent);
}

```

FIG. 5

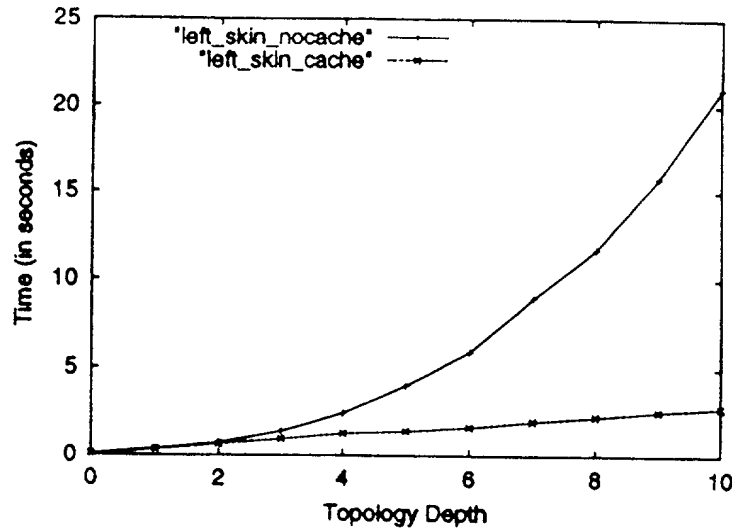
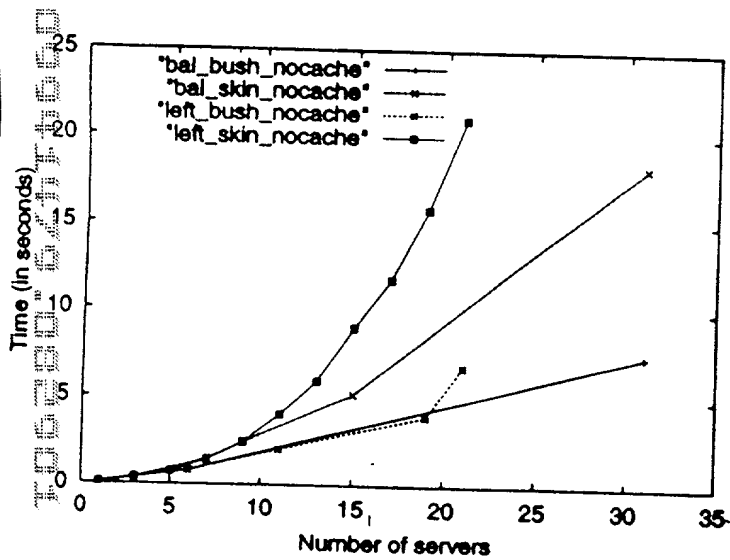
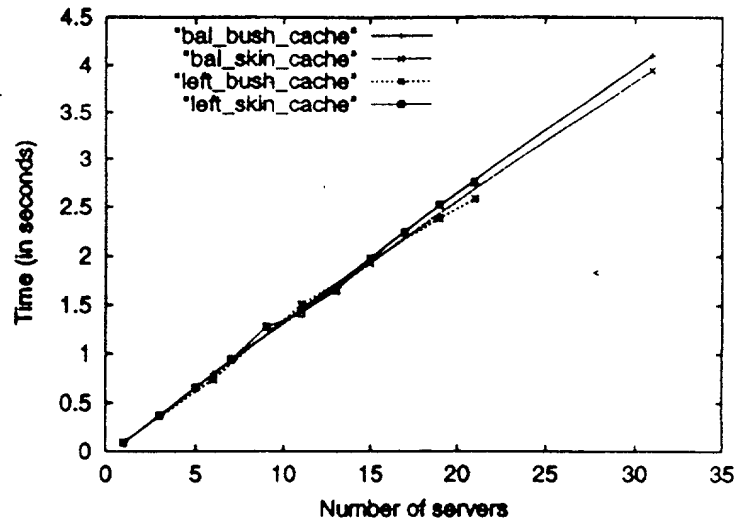


FIG. 6



(a) No Existential Cache



(b) With an Existential Cache